AF
2100
#15

PATENT **Docket No. RSW9-99-119**

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

INVENTORS:	**Kenneth McClamroch** **Avi Yaeli** **Gobi Zodik**	Examiner: C. Nguyen Art Unit: 2171
APPLICATION NO.	**09/473,554**	
FILED:	**December 29, 1999**	
TITLE:	**ASSET LOCATOR**	

CERTIFICATE OF MAIL

I hereby certify that this paper is being deposited with the U.S. Postal Service as First Class Mail, postage prepaid, in an envelope addressed to Commissioner for Patents, MAIL STOP AF, P.O. Box 1450, Alexandria, VA 22313-1450, Attention: Board of Patent Appeals and Interferences on August 25, 2003.

Lynn M. White

Commissioner for Patents
MAIL STOP AF
P.O. Box 1450
Alexandria, VA 22313-1450

Attention: Board of Patent Appeals and Interferences

APPELLANTS' BRIEF

This brief is in furtherance of the Notice of Appeal filed in this case on June 20, 2003.

This brief is transmitted in triplicate.

1. REQUIRED FEE

The requisite fee ($320.00) set forth in §1.17(f) is authorized to be charged to International Business Machines Corporation's Deposit Account No. 09-0461.

2. REAL PARTY IN INTEREST

The present application is assigned to International Business Machines Corporation, having its principal place of business at New Orchard Road, Armonk, New York. Accordingly, International Business Machines Corporation is the real party in interest.

3. RELATED APPEALS AND INTERFERENCES

The appellant, assignee, and the legal representatives of both are unaware of any other appeal or interference which will directly affect or be directly affected by or have a bearing on the Board's decision in this appeal.

4. STATUS OF CLAIMS

A. Claims canceled: None

B. Claims withdrawn from consideration but not canceled: None

C. Claims pending: 1-17

D. Claims allowed: None

E. Claims rejected: 1-17

F. Claims appealed: 1-17

Appealed claims 1-17 as currently pending are attached as Appendix A hereto.

5. STATUS OF AMENDMENTS

No amendment after final was filed in the present case. A Reply under 37 C.F.R. §1.112 was filed on February 4, 2003 and resulted in the final Office Action appealed herein.

6. SUMMARY OF THE CLAIMED INVENTION

The present invention is an asset locator (search engine) for locating software assets, code assets and the like that are stored in code repositories used by software designers. It provides the capability for the gathering of information about assets contained in the code repositories and the capturing of the gathered information in a database that can be used for the conducting of subsequent searches. The present invention has particular application in a software-development environment where the stored code assets may number in the millions and may be written in diverse languages such as, for example, Java, C/C++, COBOL, HTML, and/or XML.

A **crawl process** is performed on a storage device on which assets are stored to identify the assets. Asset-specific parameters related to the stored assets are identified, and the assets are then analyzed based upon these parameters. Textual and semantic information is extracted from the stored assets and then the extracted textual and semantic information is stored and indexed for retrieval.

In a preferred embodiment, a series of data analyzers that are specific to each type of data contained in the code repositories (e.g., a Java analyzer, a C/C++ analyzer, a COBOL analyzer, an HTML analyzer, and/or an XML analyzer) are integrated into the system so that they can be used to **crawl** the code repositories using particular attributes specific to the semantics of a particular language used to code the asset. In another preferred embodiment, the repositories are **crawled** automatically according to a schedule defined by the user, and the results of the **crawling** are stored in a database. Ordinary keyword searching can then be used with the system, either independently or combined with the attribute-specific semantic searching, to search the database.

7. ISSUES

A. ISSUE INVOLVING CLAIMS 1-17

1. Whether the Examiner improperly rejected the claims because the cited prior art fails to teach or suggest the performance of a crawl process to identify stored assets and the analysis of and extraction of information from the stored assets identified during the crawl process for storing and indexing.

8. GROUPING OF CLAIMS

A. Claims 1-17 stand or fall together

9. ARGUMENT

A. The Combination of Rose and Bowman-Anuah Fails to Teach or Suggest the Performance of a Crawl Process to Identify Stored Assets and the Analysis and Extraction of Information from the Stored Assets Identified During the Crawl Process for Storing and Indexing

The Rose Reference

U.S. Patent No. 5,752,244 to Rose et al. ("Rose") teaches a method, apparatus, and article of manufacture providing computerized management of multimedia assets of various types, including image, video, audio, text, and program code media types. Multimedia assets are checked into a computer system along with specified characteristics and identification information for the assets. Each asset is categorized according to a plurality of asset characteristics and asset identification information using predefined fields for the characteristics and identification information. In this manner, common characteristic and identification information is available for all files regardless of their type. The information to be searched is input manually and the resulting database is searched using web browser 46.

C. Bowman-Anuah

U.S. Patent No. 6,256,773 to Bowman-Anuah teaches a system, method and article of manufacture for affording consistency in a development architecture framework as components in the framework change. The Examiner relies upon Bowman-Anuah solely for the disclosure in column 4, lines 22-29, repeated herein in its entirety:

> "OOP allows the programmer to create an object that is a part of another object. For example, the object representing a piston engine is said to have

a composition-relationship with the object representing a piston. In reality, a piston engine comprises a piston, valves and many other components; the fact that a piston is an element of a piston engine can be logically and semantically represented in OOP by two objects."

B. The Combination of Rose and Bowman-Anuah Fails to Teach or Suggest the Performance of a Crawl Process to Identify Stored Assets and the Analysis and Extraction of Information from the Stored Assets Identified During the Crawl Process for Storing and Indexing

The real issue in this Appeal is a dispute over the definition of "web crawler" vs. "web browser" and the function of each. The Examiner points to web **browser** 46 of Rose and asserts that there is no distinction between a web browser and a crawl process. Applicant has provided dictionary definitions of the two terms which clearly illustrate and define the distinctions between a web crawler and a web browser. Specifically, a web crawler is used to build a database; a web browser is used to fetch a requested web page, i.e., it will search and find a specific URL input thereto, but it will not crawl a storage device, database or series of databases looking for and/or building a database consisting of the search terms. See pages 682 and 720 of "Computer Networks", 3d edition, by Andrew S. Tanenbaum (copyright 1996 by Prentiss Hall PTR), copies of which are attached hereto.

In other words, a web browser operates similar to a telephone system, in that if a user inputs a specific telephone number, the telephone system makes a connection to that number so the user may speak with the person who answers the telephone at that number. The telephone system cannot be used, however, to crawl through all telephone numbers and compile and organize them (e.g., sorting them by exchange and first letter of the last

name). The telephone terminal is simply able to take an input instruction (e.g., an input telephone number) and make a connection to the number identified by the instruction.

Similar to a telephone set, a web browser simply looks for a requested page. By contrast, a web crawler compiles and organizes entries for a search engine by reading web pages (or other data in a storage device) and related information. From the list compiled by the web crawler, a user of a web browser can select one of the specific entries identified by the web crawler to be viewed by the web browser. Like the telephone system, the web browser can simply make a connection to a URL identified by the user and/or by a web crawler. They are certainly not synonymous. Quite simply, Rose discloses the use of a web browser and makes no disclosure or suggestion of the use of a web crawler as identified in the pending claims. Further, the inclusion of Bowman-Anuah provides no such teaching or suggestion. Accordingly, the present invention, as claimed, patentably defines over these two references both alone and in combination. Accordingly, the claims should be allowed.

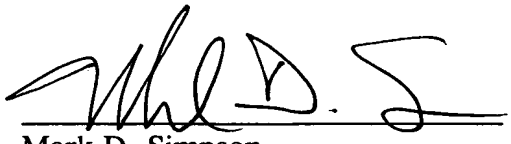
In the July 14, 2003 Advisory Action, the Examiner also states that "the claims does (sic) not crawl a database or series of databases as Applicant's arguments (sic)." Claim 1, the broadest method claim, identifies the performance of a crawl process on a storage device to identify stored assets. This is not taught or suggested by Rose or Bowman-Anuah either alone or in combination. Subsequent dependent claims specifically identify the stored assets as "assets of diverse types" (claim 2); or "code assets" (claim 4). Further, the specification clearly defines that the assets being searched, in one example, are software assets stored on servers. There is no requirement that the present invention crawl

a database *per se*, although a database is one of the types of elements upon which the crawl process can be performed.

10. CONCLUSION

Neither Rose nor Bowman-Anuah teach or suggest the use of a web crawler to crawl stored assets, identify parameters specific to those assets, analyze and store the assets based upon these parameters, extract textual and semantic information from the stored assets, and store and index the extracted textual and semantic information for retrieval. Each of these elements are found in each independent claim that is pending (and thus in each dependent claim as well). For the foregoing reasons applicants respectfully request this Board to overrule the Examiner's rejections and allow claims 1-17.

Respectfully submitted:



August 25, 2003
Date

Mark D. Simpson
Registration No. 32,942

Synnestvedt & Lechner LLP
2600 Aramark Tower
1101 Market Street
Philadelphia, PA 19107
Telephone: 215-923-4466
Facsimile: 215-923-2189

APPENDIX A

CLAIMS INVOLVED IN THIS APPEAL:

1. A computer-implemented method for indexing and locating assets stored on a storage device, comprising the steps of:

performing a crawl process on said storage device to identify stored assets;

identifying asset-specific parameters related to said stored assets;

analyzing said stored assets based on said identified asset-specific parameters;

extracting textual and semantic information from said stored assets; and

storing and indexing said extracted textual and semantic information for retrieval.

2. The method as described in claim 1, wherein said stored assets comprise assets of diverse types, and wherein said identifying step identifies the asset type of each stored asset.

3. The method as described in claim 2, wherein said extracting step includes the extraction of semantic information specific to the asset type of each stored asset.

4. The method as described in claim 3, wherein said stored assets comprise code assets and wherein said asset-specific parameters comprise languages in which each code asset is written.

5. The method as described in claim 4, wherein said analysis step is performed using language-specific analyzers corresponding to the languages of said code assets.

6. The method as described in claim 5, wherein said language-specific analyzers analyze said stored assets based on predetermined parameters specific to the language to which they correspond.

7. An indexing and locating system for indexing and locating assets stored on a storage device, comprising:

crawling means for reading the contents of said storage device to identify stored assets;

analyzing means for identifying asset-specific parameters related to said stored assets, analyzing said stored assets based on said identified asset-specific parameters, and extracting textual and semantic information from said stored assets based on said analysis; and

storing and indexing means for storing and indexing said extracted textual and semantic information for retrieval.

8. The system as set forth in claim 7, further comprising:

locating means for locating stored assets by applying a search query to said semantic information stored in said storing and indexing means.

9. The system as set forth in claim 8, wherein said locating means includes means for applying a search query to said textual information stored in said storing and indexing means.

10. The system as set forth in claim 9, wherein said locating means includes means for applying a search query to both said semantic information and said textual information simultaneously.

11. The system as set forth in claim 7, wherein said stored assets comprise assets of diverse types and wherein said analyzing means comprises an analysis server connected between said crawling means and said storing and indexing means, said analysis server including one or more asset-type specific servers, with at least one of said asset types having a corresponding asset-type specific analyzer.

12. The system as set forth in claim 11, wherein a plurality of said asset types have a corresponding asset-type specific analyzer.

13. The system as set forth in claim 12, wherein each of said asset types has a corresponding asset-type specific analyzer.

14. The system as set forth in claim 11, wherein said asset-type specific analyzer extracts predefined semantic information specific to the asset type to which it corresponds.

15. The system as described in claim 11, wherein said stored assets comprise code assets and wherein said asset-specific parameters comprise languages in which each code asset is written.

16. The system as described in claim 15, wherein said analysis step is performed using language-specific analyzers corresponding to the languages of said code assets.

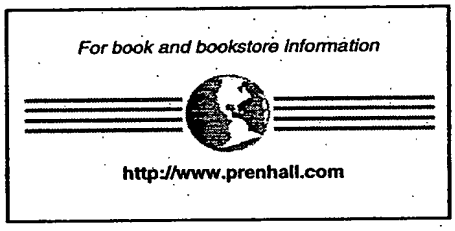
17. The system as described in claim 16, wherein said language-specific analyzers analyze said stored assets based on predetermined parameters specific to the language to which they correspond.

Computer Networks

Third Edition

Andrew S. Tanenbaum

Vrije Universiteit
Amsterdam, The Netherlands



Prentice Hall PTR
Upper Saddle River, New Jersey 07458

system, we will discuss both the client (i.e., user) side and the server side. Then we will examine the language in which Web pages are written (HTML and Java). Finally, comes an examination of how to find information on the Web.

7.6.1. The Client Side

From the users' point of view, the Web consists of a vast, worldwide collection of documents, usually just called **pages** for short. Each page may contain links (pointers) to other, related pages, anywhere in the world. Users can follow a link (e.g., by clicking on it), which then takes them to the page pointed to. This process can be repeated indefinitely, possibly traversing hundreds of linked pages while doing so. Pages that point to other pages are said to use **hypertext.**

Pages are viewed with a program called a **browser**, of which Mosaic and Netscape are two popular ones. The browser fetches the page requested, interprets the text and formatting commands that it contains, and displays the page, properly formatted, on the screen. An example is given in Fig. 7-58(a). Like many Web pages, this one starts with a title, contains some information, and ends with the email address of the page's maintainer. Strings of text that are links to other pages, called **hyperlinks**, are highlighted, either by underlining, displaying them in a special color, or both. To follow a link, the user places the cursor on the highlighted area (using the mouse or the arrow keys) and selects it (by clicking a mouse button or hitting ENTER). Although nongraphical browsers, such as Lynx, exist, they are not as popular as graphical browsers, so we will concentrate on the latter. Voice-based browsers are also being developed.

Users who are curious about the Department of Animal Psychology can learn more about it by clicking on its (underlined) name. The browser then fetches the page to which the name is linked and displays it, as shown in Fig. 7-58(b). The underlined items here can also be clicked on to fetch other pages, and so on. The new page can be on the same machine as the first one, or on a machine halfway around the globe. The user cannot tell. Page fetching is done by the browser, without any help from the user. If the user ever returns to the main page, the links that have already been followed may be shown with a dotted underline (and possibly a different color) to distinguish them from links that have not been followed. Note that clicking on the *Campus Information* line in the main page does nothing. It is not underlined, which means that it is just text and is not linked to another page.

Most browsers have numerous buttons and features to make it easier to navigate the Web. Many have a button for going back to the previous page, a button for going forward to the next page (only operative after the user has gone back from it), and a button for going straight to the user's own home page. Most browsers have a button or menu item to set a bookmark on a given page and another one to display the list of bookmarks, making it possible to revisit any of

bytes, individual frames can use a strong error-correcting code, and all frames can be sent two or three times. Many other covert channels exist, and it is extremely difficult to discover and block them all. For more information about the security problems in Java see (Dean and Wallach, 1995).

In short, Java introduces many new possibilities and opportunities into the World Wide Web. It allows Web pages to be interactive, and to contain animation and sound. It also permits browsers to be infinitely extensible. However, the Java model of downloading applets also introduces some serious new security problems that have not been entirely solved yet.

7.6.5. Locating Information on the Web

Although the Web contains a vast amount of information, finding the right item is not always easy. To make it easier for people to find pages that are useful to them, several researchers have written programs to index the Web in various ways. Some of these have become so popular that they have gone commercial. Programs that search the Web are sometimes called **search engines**, **spiders**, **crawlers**, **worms**, or **knowbots** (knowledge robots). In this section we will give a brief introduction to this subject. For more information, see (Pinkerton, 1994; and McBryan, 1994).

Although the Web is huge, reduced to its barest essentials, the Web is a big graph, with the pages being the nodes and the hyperlinks being the arcs. Algorithms for visiting all the nodes in a graph are well known. What makes Web indexing difficult is the enormous amount of data that must be managed and the fact that it is constantly changing.

Let us start our discussion with a simple goal: indexing all the keywords in Web pages' titles. For our algorithm, we will need three data structures. First, we need a large, linear array, *url_table*, that contains millions of entries, ultimately one per Web page. It should be kept in virtual memory, so parts not heavily used will automatically be paged to disk. Each entry contains two pointers, one to the page's URL and one to the page's title. Both of these items are variable length strings and can be kept on a heap (a large unstructured chunk of virtual memory to which strings can be appended). The heap is our second data structure.

The third data structure is a hash table of size n entries. It is used as follows. Any URL can be run through a hash function to produce a nonnegative integer less than n. All URLs that hash to the value k are hooked together on a linked list starting at entry k of the hash table. Whenever a URL is entered into *url_table*, it is also entered into the hash table. The main use of the hash table is to start with a URL and be able to quickly determine whether it is already present in *url_table*. These three data structures are illustrated in Fig. 7-75.

Building the index requires two phases: searching and indexing. Let us start with a simple engine for doing the searching. The heart of the search engine is a recursive procedure *process_url*, which takes a URL string as input. It operates as